SENSIRION
THE SENSOR COMPANY

Application Note for I²C Flow and Differential Pressure Sensors
Sample Code

Summary

Sensirion's I²C flow and differential pressure sensors are based on a 4th generation CMOSens® mass flow sensor chip called SF04. The sensors can directly be connected to a microcontroller. This application note contains a C++ sample code to implement the basic commands.

1. Sample Code

Due to compatibility reasons the I²C interface is implemented as "bit-banging" on normal I/O's. This code is written for an easy understanding and is neither optimized for speed nor code size. A copy of the code may be found on the following pages of this application note.

The source code is available as zip archive on our webpage or from Sensirion by request.

2. Revision history

Date	Version	Changes
July 2009	v1.0	Initial release

Headquarter and Sales Offices

SENSIRION AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland
Phone: + 41 (0)44 306 40 00
Fax: + 41 (0)44 306 40 30
info@sensirion.com
www.sensirion.com

SENSIRION Inc
Westlake Pl. Ctr. I, suite 204
2801 Townsgate Road
Westlake Village, CA 91361
USA
Phone: +1 805-409 4900
Fax: +1 805-435 0467
michael.karst@sensirion.com
www.sensirion.com

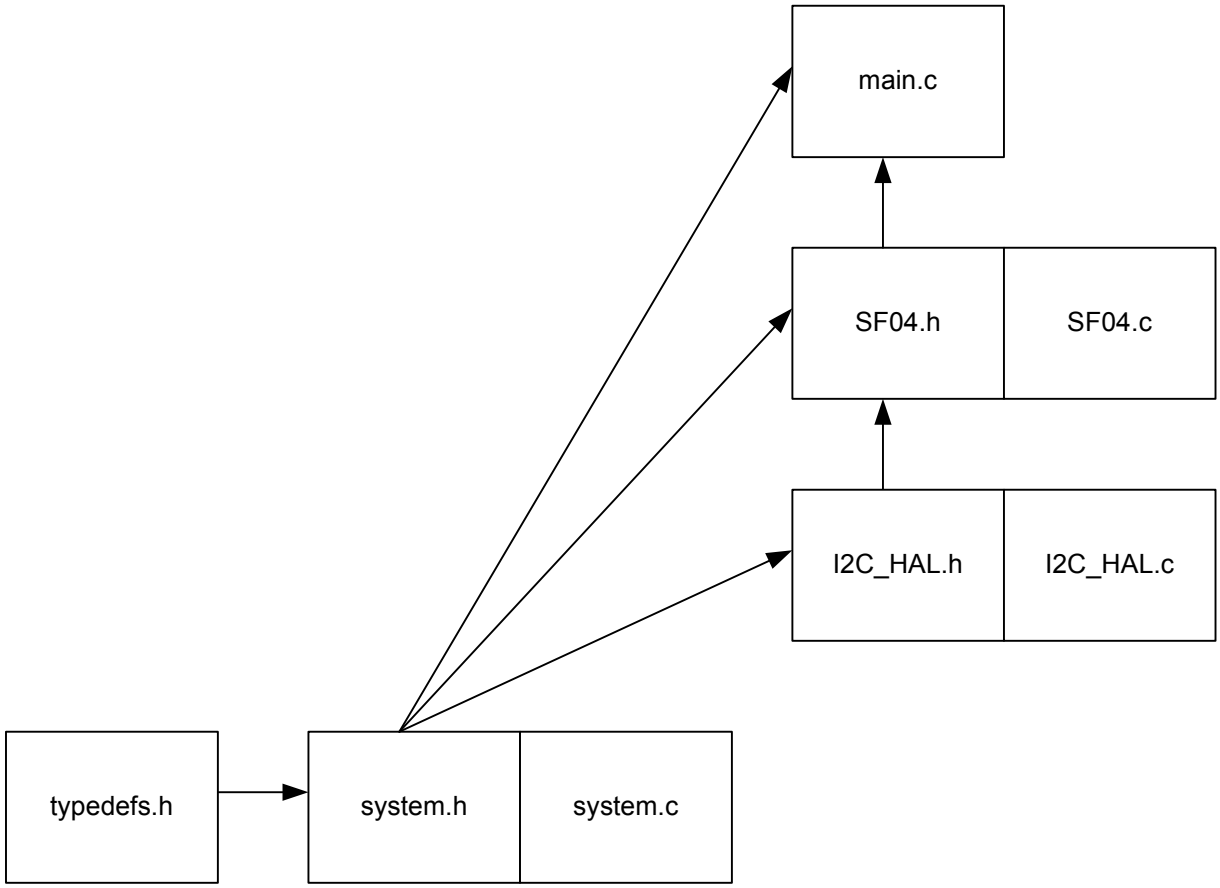
SENSIRION Japan
Sensirion Japan Co. Ltd.
Shinagawa Station Bldg. 7F
4-23-5 Takanawa
Minato-ku, Tokyo, Japan
phone: +81 3-3444-4940
fax: +81 3-3444-4939
info@sensirion.co.jp
www.sensirion.co.jp

SENSIRION Korea Co. Ltd.
#1414, Anyang Construction Tower B/D,
1112-1, Bisan-dong, Anyang-city,
Gyeonggi-Province, South Korea
Phone: +82-31-440-9925~27
Fax: +82-31-440-9927
info@sensirion.co.kr
www.sensirion.co.kr

SENSIRION China Co. Ltd.
Room 2411, Main Tower
Jin Zhong Huan Business Building,
Postal Code 518048
Futian District, Shenzhen, PR China
Phone: +86 755 8252 1501
Fax: +86 755 8252 1580
info@sensirion.com.cn/
www.sensirion.com.cn

Find your local representative at: http://www.sensirion.com/reps

Structure SF04 Sample Code



Revision History

Revision	Date	Changes
V1.0	27.4.09 / MST	Initial draft

```
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  main.h
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  This code shows how to implement the basic commands for a
//              flow or differential pressure senor based on SF04 sensor chip.
//              Due to compatibility reasons the I2C interface is implemented
//              as "bit-banging" on normal I/O's. This code is written for an
//              easy understanding and is neither optimized for speed nor code
//              size.
//
// Porting to a different microcontroller (uC):
//   - define the byte-order for your uC (e.g. little endian) in typedefs.h
//   - the definitions of basic types may have to be changed  in typedefs.h
//   - change the port functions / definitions for your uC    in I2C_HAL.h/.c
//   - adapt the timing of the delay function for your uC     in system.c
//   - adapt the HW_Init()                                    in system.c
//   - change the uC register definition file <io70f3740.h>   in system.h
//---------- Includes ---------------------------------------------------------
#include "SF04.h"
#include "System.h"
#include <stdio.h>
//===========================================================================
int main()
//===========================================================================
{  // variables
  u8t error = 0;         //variable for error code. For error codes see system.h
  nt16 registerValue;    //variable for a sensor's register value
  nt16 measurand;        //variable for a sensor measurement
  nt32 snProduct;        //variable for the sensor's product serial number
  nt16 scaleFactor;      //variable for the sensor's scale factor
  ft flow;               //variable for measured flow as a float value
  char flowUnitStr[15];  //string for the flow unit
  char output[40];       //output string for measured value and unit

  Init_HW();                  //Initializes the Hardware (osc, watchdog,...)
  I2c_Init();                 //Initializes the uC-ports for I2C
  DelayMicroSeconds(20000);   //wait for sensor initialization t_INIT (20ms)
  //-- get product serial number and convert it to a string--
  error |= SF04_ReadSerialNumber(&snProduct);
  sprintf(output, "Product Serial Number: %ld \n",snProduct.u32);
  //-- get scale factor flow --
  error |= SF04_ReadScaleFactor(&scaleFactor);
  //-- get flow unit --
  error |= SF04_ReadFlowUnit(flowUnitStr);
  //-- set measurement resolution to 14 bit --
  error |= SF04_ReadRegister(ADV_USER_REG,&registerValue);
  registerValue.u16 = (registerValue.u16 & ~eSF04_RES_MASK) | eSF04_RES_14BIT;
  error |= SF04_WriteRegister(ADV_USER_REG,&registerValue);
  while(1)
  { //-- measure flow and convert to a string --
    error |= SF04_Measure(FLOW, &measurand);
    //-- calculate flow --
    flow = (float)measurand.i16 / scaleFactor.u16;    //for bidirectional flow
    sprintf(output, "%f %s \n",flow,flowUnitStr);
    DelayMicroSeconds(10000);
  }
}
```

```
#ifndef SF04_H
#define SF04_H
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  SF04.h
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  Sensor layer. Definitions of commands, registers, eeprom adr.
//              functions for sensor access
//===========================================================================

//---------- Includes ---------------------------------------------------------
#include "I2C_HAL.h"
#include "system.h"
//---------- Defines ----------------------------------------------------------
//  CRC
#define POLYNOMIAL 0x131  //P(x)=x^8+x^5+x^4+1 = 100110001

//  SF04 eeprom map
#define EE_ADR_SN_CHIP       0x02E4
#define EE_ADR_SN_PRODUCT    0x02F8
#define EE_ADR_SCALE_FACTOR  0x02B6
#define EE_ADR_FLOW_UNIT     0x02B7

// sensor command
typedef enum{
  USER_REG_W               = 0xE2, // command writing user register
  USER_REG_R               = 0xE3, // command reading user register
  ADV_USER_REG_W           = 0xE4, // command writing advanced user register
  ADV_USER_REG_R           = 0xE5, // command reading advanced user register
  READ_ONLY_REG1_R         = 0xE7, // command reading read-only register 1
  READ_ONLY_REG2_R         = 0xE9, // command reading read-only register 2
  TRIGGER_FLOW_MEASUREMENT = 0xF1, // command trig. a flow measurement
  TRIGGER_TEMP_MEASUREMENT = 0xF3, // command trig. a temperature measurement
  TRIGGER_VDD_MEASUREMENT  = 0xF5, // command trig. a supply voltage measurement
  EEPROM_W                 = 0xFA, // command writing eeprom
  EEPROM_R                 = 0xFA, // command reading eeprom
  SOFT_RESET               = 0xFE  // command soft reset
}etCommand;

// sensor register
typedef enum{
  USER_REG                 = USER_REG_R,
  ADV_USER_REG             = ADV_USER_REG_R,
  READ_ONLY_REG1           = READ_ONLY_REG1_R,
  READ_ONLY_REG2           = READ_ONLY_REG2_R
}etSF04Register;

// measurement signal selection
typedef enum{
  FLOW                     = TRIGGER_FLOW_MEASUREMENT,
  TEMP                     = TRIGGER_TEMP_MEASUREMENT,
  VDD                      = TRIGGER_VDD_MEASUREMENT,
}etSF04MeasureType;

// This enum lists all available flow resolution (Advanced User Register [11:9])
typedef enum {
  eSF04_RES_9BIT                 = ( 0<<9  ),
  eSF04_RES_10BIT                = ( 1<<9  ),
```

```
  eSF04_RES_11BIT                = ( 2<<9  ),
  eSF04_RES_12BIT                = ( 3<<9  ),
  eSF04_RES_13BIT                = ( 4<<9  ),
  eSF04_RES_14BIT                = ( 5<<9  ),
  eSF04_RES_15BIT                = ( 6<<9  ),
  eSF04_RES_16BIT                = ( 7<<9  ),
  eSF04_RES_MASK                 = ( 7<<9  )
} etSF04Resolution;

//===========================================================================
u8t SF04_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum);
//===========================================================================
// calculates checksum for n bytes of data and compares it with expected
// checksum
// input:  data[]       checksum is built based on this data
//         nbrOfBytes   checksum is built for n bytes of data
//         checksum     expected checksum
// return: error:       CHECKSUM_ERROR = checksum does not match
//                      0              = checksum matches

//===========================================================================
u8t SF04_ReadRegister(etSF04Register eSF04Register, nt16 *pRegisterValue);
//===========================================================================
// reads the selected SF04 register (16bit)
// input : eSF04Register register selection
// output: *pRegisterValue
// return: error

//===========================================================================
u8t SF04_WriteRegister(etSF04Register eSF04Register, nt16 *pRegisterValue);
//===========================================================================
// writes the selected SF04 register (16bit)
// input:  eSF04Register register selection
//         *pRegisterValue
// output: -
// return: error

//===========================================================================
u8t SF04_ReadEeprom( u16t eepromStartAdr, u16t size, nt16 eepromData[]);
//===========================================================================
// reads data from the SF04's eeprom
// input : eepromStartAdr : Eeprom address of first word to read
//         size           : number of words(16bit) to read
// output: eepromData[]   : pointer to a u16t array
// return: error

//===========================================================================
u8t SF04_Measure(etSF04MeasureType eSF04MeasureType, nt16 *pMeasurand);
//===========================================================================
// measures flow, temperature, VDD
// input:  eSF04MeasureType
// output: *pMeasurand
// return: error
// note:   timing for timeout may be changed

//===========================================================================
u8t SF04_ReadSerialNumber( nt32 *serialNumber );
//===========================================================================
// reads the product's serial number
// input:  -
// output: *serialNumber: the product's serial number
// return: error
```

```
//===========================================================================
u8t SF04_ReadScaleFactor(nt16 *scaleFactor);
//===========================================================================
// reads the scale factor of the active calibration field
// input:  -
// output: *scaleFactor: scale facor of active calibration field
// return: error

//===========================================================================
u8t SF04_ReadFlowUnit(char *flowUnitStr);
//===========================================================================
// reads the flow unit of the active calibration field
// input:  -
// output: *flowUnitStr: pointer to unit string
// return: error
// note: flowUnitStr may be up to 11 characters (incl. string termination 0x00)

#endif
```

```
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  SF04.cpp
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  Sensor layer. Definitions of commands, registers, eeprom adr.
//              functions for sensor access
//===========================================================================

//---------- Includes -------------------------------------------------------
#include "SF04.h"
#include <string.h>

//===========================================================================
u8t SF04_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum)
//===========================================================================
{
  u8t crc = 0;
  u8t byteCtr;
  //calculates 8-Bit checksum with given polynomial
  for (byteCtr = 0; byteCtr < nbrOfBytes; ++byteCtr)
  { crc ^= (data[byteCtr]);
    for (u8t bit = 8; bit > 0; --bit)
    { if (crc & 0x80) crc = (crc << 1) ^ POLYNOMIAL;
      else crc = (crc << 1);
    }
  }
  if (crc != checksum) return CHECKSUM_ERROR;
  else return 0;
}

//===========================================================================
u8t SF04_ReadRegister(etSF04Register eSF04Register, nt16 *pRegisterValue)
//===========================================================================
{
  u8t checksum;   //variable for checksum byte
  u8t data[2];    //data array for checksum verification
  u8t error=0;    //variable for error code

  I2c_StartCondition();
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_WRITE);
  error |= I2c_WriteByte (eSF04Register);
  I2c_StartCondition();
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_READ);
  pRegisterValue->s16.u8H = data[0] = I2c_ReadByte(ACK);
  pRegisterValue->s16.u8L = data[1] = I2c_ReadByte(ACK);
  checksum=I2c_ReadByte(NO_ACK);
  error |= SF04_CheckCrc (data,2,checksum);
  I2c_StopCondition();
  return error;
}

//===========================================================================
u8t SF04_WriteRegister(etSF04Register eSF04Register, nt16 *pRegisterValue)
//===========================================================================
{
  u8t error=0;   //variable for error code

  //-- check if selected register is writable --
```

```
  assert(!(eSF04Register == READ_ONLY_REG1 || eSF04Register == READ_ONLY_REG2));
  //-- write register to sensor --
  I2c_StartCondition();
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_WRITE);
  error |= I2c_WriteByte (eSF04Register & ~I2C_RW_MASK | I2C_WRITE);
  error |= I2c_WriteByte (pRegisterValue->s16.u8H);
  error |= I2c_WriteByte (pRegisterValue->s16.u8L);
  I2c_StopCondition();
  return error;
}

//==============================================================================
u8t SF04_ReadEeprom( u16t eepromStartAdr, u16t size, nt16 eepromData[])
//==============================================================================
{
  u8t  checksum;   //checksum
  u8t  data[2];    //data array for checksum verification
  u8t  error=0;    //error variable
  u16t i;          //counting variable

  nt16 eepromStartAdrTmp;  //variable for eeprom adr. as nt16
  eepromStartAdrTmp.u16=eepromStartAdr;

  //-- write I2C sensor address and command --
  I2c_StartCondition();
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_WRITE);
  error |= I2c_WriteByte (EEPROM_R);
  //-- write 12-bit eeprom address left aligned --
  eepromStartAdrTmp.u16=(eepromStartAdrTmp.u16<<4);   // align eeprom adr left
  error |= I2c_WriteByte (eepromStartAdrTmp.s16.u8H);
  error |= I2c_WriteByte (eepromStartAdrTmp.s16.u8L);
  //-- write I2C sensor address for read --
  I2c_StartCondition();
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_READ);
  //-- read eeprom data and verify checksum --
  for(i=0; i<size; i++)
  { eepromData[i].s16.u8H = data[0] = I2c_ReadByte(ACK);
    eepromData[i].s16.u8L = data[1] = I2c_ReadByte(ACK);
    checksum=I2c_ReadByte( (i < size-1) ? ACK : NO_ACK ); //NACK for last byte
    error |= SF04_CheckCrc (data,2,checksum);
  }
  I2c_StopCondition();
  return error;
}

//==============================================================================
u8t SF04_Measure(etSF04MeasureType eSF04MeasureType, nt16 *pMeasurand)
//==============================================================================
{
  u8t  checksum;   //checksum
  u8t  data[2];    //data array for checksum verification
  u8t  error=0;    //error variable
  u16t i;          //counting variable

  I2c_StartCondition();
  //-- write I2C sensor address + write bit --
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_WRITE);
  //-- write measurement command --
  switch(eSF04MeasureType)
  { case FLOW :  error |= I2c_WriteByte (TRIGGER_FLOW_MEASUREMENT); break;
    case TEMP :  error |= I2c_WriteByte (TRIGGER_TEMP_MEASUREMENT); break;
    case VDD  :  error |= I2c_WriteByte (TRIGGER_VDD_MEASUREMENT) ; break;
```

```
    default: assert(0);
  }
  I2c_StartCondition();
  //-- write I2C sensor address + read bit--
  error |= I2c_WriteByte (I2C_ADR<<1 & ~I2C_RW_MASK | I2C_READ);
  //-- wait until hold master is released --
  SCL=HIGH;                     // set SCL I/O port as input
  for(i=0; i<1000; i++)         // wait until master hold is released or
  { DelayMicroSeconds(1000);    // a timeout (~1s) is reached
    if (SCL_CONF==1) break;
  }
  //-- check for timeout --
  if(SCL_CONF==0) error |= TIME_OUT_ERROR;
  //-- read two data bytes and one checksum byte --
  pMeasurand->s16.u8H = data[0] = I2c_ReadByte(ACK);
  pMeasurand->s16.u8L = data[1] = I2c_ReadByte(ACK);
  checksum=I2c_ReadByte(NO_ACK);
  //-- verify checksum --
  error |= SF04_CheckCrc (data,2,checksum);
  I2c_StopCondition();
  return error;
}

//===========================================================================
u8t SF04_ReadSerialNumber( nt32 *serialNumber )
//===========================================================================
{
  nt16 registerValue;  //register value for register
  u16t eepromBaseAdr;  //eeprom base address of active calibration field
  u16t eepromAdr;      //eeprom address of SF04's scale factor
  nt16 eepromData[2];  //serial number
  u8t  error=0;        //error variable

  //-- read "Read-Only Register 2" to find out the active configuration field --
  error |= SF04_ReadRegister(READ_ONLY_REG2,&registerValue);
  //-- calculate eeprom address of product serial number --
  eepromBaseAdr=(registerValue.u16 & 0x0007)*0x0300; //RO_REG2 bit<2:0>*0x0300
  eepromAdr= eepromBaseAdr + EE_ADR_SN_PRODUCT;
  //-- read product serial number from SF04's eeprom--
  error |= SF04_ReadEeprom( eepromAdr, 2, eepromData);
  serialNumber->s32.u16H=eepromData[0].u16;
  serialNumber->s32.u16L=eepromData[1].u16;
  return error;
}

//===========================================================================
u8t SF04_ReadScaleFactor(nt16 *scaleFactor)
//===========================================================================
{
  nt16 registerValue;     //register value for user register
  u16t eepromBaseAdr;     //eeprom base address of active calibration field
  u16t eepromAdr;         //eeprom address of SF04's scale factor
  u8t  error=0;           //error variable

  //-- read "User Register " to find out the active calibration field --
  error |= SF04_ReadRegister(USER_REG ,&registerValue);
  //-- calculate eeprom address of scale factor --
  eepromBaseAdr=((registerValue.u16 & 0x0070)>>4)*0x0300; //UserReg bit<6:4>*0x0300
  eepromAdr= eepromBaseAdr + EE_ADR_SCALE_FACTOR;
  //-- read scale factor from SF04's eeprom--
  error |= SF04_ReadEeprom( eepromAdr, 1, scaleFactor);
```

```
  return error;
}

//===========================================================================
u8t SF04_ReadFlowUnit(char *flowUnitStr)
//===========================================================================
{
  //-- table for unit dimension, unit time, unit volume (x=not defined) --
  const char *unitDimension[]={"x","x","x","n","u","m","c","d","","-","h","k",
                               "M","G","x","x"};
  const char *unitTimeBase[] ={"","us","ms","s","min","h","day","x","x","x","x",
                               "x","x","x","x","x"};
  const char *unitVolume[]   ={"ln","sl","x","x","x","x","x","x","l","g","x",
                               "x","x","x","x","x","Pa","bar","mH2O","inH2O",
                               "x","x","x","x","x","x","x","x","x","x","x","x"};
  //-- local variables --
  nt16 registerValue;     //register value for  user register
  u16t eepromBaseAdr;     //eeprom base address of active calibration field
  u16t eepromAdr;         //eeprom address of SF04's flow unit word
  nt16 flowUnit;          //content of SF04's flow unit word
  u8t  tableIndex;        //index of one of the unit arrays
  u8t  error=0;

  //-- read "User Register" to find out the active calibration field --
  error |= SF04_ReadRegister(USER_REG ,&registerValue);
  //-- calculate eeprom address of flow unit--
  eepromBaseAdr=((registerValue.u16 & 0x0070)>>4)*0x0300; //UserReg bit<6:4>*0x0300
  eepromAdr= eepromBaseAdr + EE_ADR_FLOW_UNIT;
  //-- read flow unit from SF04's eeprom--
  error |= SF04_ReadEeprom( eepromAdr, 1, &flowUnit);
  //-- get index of corresponding table and copy it to unit string --
  tableIndex=(flowUnit.u16 & 0x000F)>>0;         //flowUnit bit <3:0>
  strcpy(flowUnitStr, unitDimension[tableIndex]);
  tableIndex=(flowUnit.u16 & 0x1F00)>>8;         //flowUnit bit <8:12>
  strcat(flowUnitStr, unitVolume[tableIndex]);
  tableIndex=(flowUnit.u16 & 0x00F0)>>4;         //flowUnit bit <4:7>
  if(unitTimeBase[tableIndex] != "")             //check if time base is defined
  { strcat(flowUnitStr, "/");
    strcat(flowUnitStr, unitTimeBase[tableIndex]);
  }
  //-- check if unit string is feasible --
  if(strchr(flowUnitStr,'x') != NULL ) error |= UNIT_ERROR;

  return error;
}
```

```
#ifndef I2C_HAL_H
#define I2C_HAL_H
//=============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//=============================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  I2C_HAL.h
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  I2C Hardware abstraction layer
//=============================================================================

//---------- Includes ----------------------------------------------------------
#include "system.h"

//---------- Defines -----------------------------------------------------------
//I2C ports
//The communication on SDA and SCL is done by switching pad direction
//For a low level on SCL or SDA, direction is set to output. For a high level on
//SCL or SDA, direction is set to input. (pull up resistor active)
#define SDA      PM3H_bit.no0 //SDA on I/O P38 defines direction (input=1/output=0)
#define SDA_CONF P3H_bit.no0  //SDA level on output direction
#define SCL      PM3H_bit.no1 //SCL on I/O P39 defines direction (input=1/output=0)
#define SCL_CONF P3H_bit.no1  //SCL level on output direction

//---------- Enumerations ------------------------------------------------------
//  I2C header
typedef enum{
  I2C_ADR                 = 64,   // default sensor I2C address
  I2C_WRITE               = 0x00, // write bit in header
  I2C_READ                = 0x01, // read bit in header
  I2C_RW_MASK             = 0x01  // bit position of read/write bit in header
}etI2cHeader;

//  I2C level
typedef enum{
  LOW                     = 0,
  HIGH                    = 1,
}etI2cLevel;

// I2C acknowledge
typedef enum{
  ACK                     = 0,
  NO_ACK                  = 1,
}etI2cAck;

//=============================================================================
void I2c_Init ();
//=============================================================================
//Initializes the ports for I2C interface

//=============================================================================
void I2c_StartCondition ();
//=============================================================================
// writes a start condition on I2C-bus
// input : -
// output: -
// return: -
// note  : timings (delay) may have to be changed for different microcontroller
//       _____
// SDA:       |_____
```

```
//       _______
// SCL :        |___

//=========================================================================
void I2c_StopCondition ();
//=========================================================================
// writes a stop condition on I2C-bus
// input : -
// output: -
// return: -
// note  : timings (delay) may have to be changed for different microcontroller
//              _____
// SDA:   _____|
//           _______
// SCL :  ___|

//=========================================================================
u8t I2c_WriteByte (u8t txByte);
//=========================================================================
// writes a byte to I2C-bus and checks acknowledge
// input:  txByte  transmit byte
// output: -
// return: error
// note: timings (delay) may have to be changed for different microcontroller

//=========================================================================
u8t I2c_ReadByte (etI2cAck ack);
//=========================================================================
// reads a byte on I2C-bus
// input:  rxByte  receive byte
// output: rxByte
// note: timings (delay) may have to be changed for different microcontroller

#endif
```

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  I2C_HAL.cpp
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  I2C Hardware abstraction layer
//==============================================================================

//---------- Includes ----------------------------------------------------------
#include "I2C_HAL.h"

//==============================================================================
void I2c_Init ()
//==============================================================================
{
  SDA=LOW;                // Set port as output for configuration
  SCL=LOW;                // Set port as output for configuration

  SDA_CONF=LOW;           // Set SDA level as low for output mode
  SCL_CONF=LOW;           // Set SCL level as low for output mode

  SDA=HIGH;               // I2C-bus idle mode SDA released (input)
  SCL=HIGH;               // I2C-bus idle mode SCL released (input)
}

//==============================================================================
void I2c_StartCondition ()
//==============================================================================
{
  SDA=HIGH;
  SCL=HIGH;

  SDA=LOW;
  DelayMicroSeconds(10);  // hold time start condition (t_HD;STA)
  SCL=LOW;
  DelayMicroSeconds(10);
}

//==============================================================================
void I2c_StopCondition ()
//==============================================================================
{
  SDA=LOW;
  SCL=LOW;
  SCL=HIGH;
  DelayMicroSeconds(10);  // set-up time stop condition (t_SU;STO)
  SDA=HIGH;
  DelayMicroSeconds(10);
}

//==============================================================================
u8t I2c_WriteByte (u8t txByte)
//==============================================================================
{
  u8t mask,error=0;
  for (mask=0x80; mask>0; mask>>=1)   //shift bit for masking (8 times)
  { if ((mask & txByte) == 0) SDA=LOW;//masking txByte, write bit to SDA-Line
    else SDA=HIGH;
    DelayMicroSeconds(1);             //data set-up time (t_SU;DAT)
```

```
    SCL=HIGH;                          //generate clock pulse on SCL
    DelayMicroSeconds(5);              //SCL high time (t_HIGH)
    SCL=LOW;
    DelayMicroSeconds(1);              //data hold time(t_HD;DAT)
  }
  SDA=HIGH;                            //release SDA-line
  SCL=HIGH;                            //clk #9 for ack
  DelayMicroSeconds(1);                //data set-up time (t_SU;DAT)
  if(SDA_CONF==HIGH) error=ACK_ERROR; //check ack from i2c slave
  SCL=LOW;
  DelayMicroSeconds(20);               //wait time to see byte package on scope
  return error;                        //return error code
}

//===========================================================================
u8t I2c_ReadByte (etI2cAck ack)
//===========================================================================
{
  u8t mask,rxByte=0;
  SDA=HIGH;                            //release SDA-line
  for (mask=0x80; mask>0; mask>>=1)   //shift bit for masking (8 times)
  { SCL=HIGH;                          //start clock on SCL-line
    DelayMicroSeconds(1);              //data set-up time (t_SU;DAT)
    DelayMicroSeconds(3);              //SCL high time (t_HIGH)
    if (SDA_CONF==1) rxByte=(rxByte | mask); //read bit
    SCL=LOW;
    DelayMicroSeconds(1);              //data hold time(t_HD;DAT)
  }
  SDA=ack;                             //send acknowledge if necessary
  DelayMicroSeconds(1);                //data set-up time (t_SU;DAT)
  SCL=HIGH;                            //clk #9 for ack
  DelayMicroSeconds(5);                //SCL high time (t_HIGH)
  SCL=LOW;
  SDA=HIGH;                            //release SDA-line
  DelayMicroSeconds(20);               //wait time to see byte package on scope
  return rxByte;                       //return error code
}
```

```
#ifndef SYSTEM_H
#define SYSTEM_H
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  System.h
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  System functions, global definitions
//===========================================================================

//---------- Includes ---------------------------------------------------------
#include <io70f3740.h>             // controller register definitions
#include <assert.h>                // assert functions
#include <intrinsics.h>            // low level microcontroller commands
#include "typedefs.h"              // type definitions

//---------- Enumerations -----------------------------------------------------
// Error codes
typedef enum{
  ACK_ERROR               = 0x01,
  TIME_OUT_ERROR          = 0x02,
  CHECKSUM_ERROR          = 0x04,
  UNIT_ERROR              = 0x08
}etError;

//===========================================================================
void Init_HW (void);
//===========================================================================
// Initializes the used hardware

//===========================================================================
void DelayMicroSeconds (u32t nbrOfUs);
//===========================================================================
// wait function for small delays
// input:  nbrOfUs   wait x times approx. one micro second (fcpu = 4MHz)
// return: -
// note: smallest delay is approx. 30us due to function call

#endif
```

```
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  System.cpp
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  System functions
//===========================================================================

//---------- Includes ---------------------------------------------------------
#include "system.h"

//===========================================================================
void Init_HW (void)
//===========================================================================
{
  //-- initialize system clock of V850 (fcpu = fosc, no PLL) --
  PRCMD = 0x00;   // unlock PCC register
  PCC = 0x00;     // perform settings in PCC register
  RCM = 0x01;     // disable ring oscillator
  //-- watchdog --
  WDTM2 = 0x0f;   // stop watchdog
  //-- interrupts --
  __EI();         // enable interrupts for debugging with minicube

  //Settings for debugging with Sensirion EKH4 and minicube2, power up sensor
  //Not needed for normal use
  PMDLL = 0xF0;
  PDLL  = 0x04;
}

//===========================================================================
#pragma optimize = s none
void DelayMicroSeconds (u32t nbrOfUs)
//===========================================================================
{
  for(u32t i=0; i<nbrOfUs; i++)
  {  //__asm("nop");  //nop's may be added for timing adjustment
  }
}
```

```
#ifndef TYPEDEFS_H
#define TYPEDEFS_H
//============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//============================================================================
// Project   :  SF04 Sample Code (V1.0)
// File      :  typedefs.h
// Author    :  MST
// Controller:  NEC V850/SG3 (uPD70F3740)
// Compiler  :  IAR compiler for V850 (3.50A)
// Brief     :  Definitions of typedefs for good readability and portability
//============================================================================

//---------- Defines -----------------------------------------------------------
//Processor endian system
//#define BIG ENDIAN   //e.g. Motorola (not tested at this time)
#define LITTLE_ENDIAN  //e.g. PIC, 8051, NEC V850
//============================================================================
// basic types: making the size of types clear
//============================================================================
typedef unsigned char   u8t;      ///< range: 0 .. 255
typedef signed char     i8t;      ///< range: -128 .. +127

typedef unsigned short  u16t;     ///< range: 0 .. 65535
typedef signed short    i16t;     ///< range: -32768 .. +32767

typedef unsigned long   u32t;     ///< range: 0 .. 4'294'967'295
typedef signed long     i32t;     ///< range: -2'147'483'648 .. +2'147'483'647

typedef float           ft;       ///< range: +-1.18E-38 .. +-3.39E+38
typedef double          dt;      ///< range:            .. +-1.79E+308

typedef bool            bt;       ///< values: 0, 1 (real bool used)

typedef union {
  u16t u16;               // element specifier for accessing the whole u16
  i16t i16;               // element specifier for accessing the whole i16
  struct {
    #ifdef LITTLE_ENDIAN  // Byte-order is little endian
    u8t u8L;              // element specifier for accessing the low u8
    u8t u8H;              // element specifier for accessing the high u8
    #else                 // Byte-order is big endian
    u8t u8H;              // element specifier for accessing the low u8
    u8t u8L;              // element specifier for accessing the high u8
    #endif
  } s16;                  // element spec. for acc. struct with low or high u8
} nt16;

typedef union {
  u32t u32;               // element specifier for accessing the whole u32
  i32t i32;               // element specifier for accessing the whole i32
 struct {
    #ifdef LITTLE_ENDIAN  // Byte-order is little endian
    u16t u16L;            // element specifier for accessing the low u16
    u16t u16H;            // element specifier for accessing the high u16
    #else                 // Byte-order is big endian
    u16t u16H;            // element specifier for accessing the low u16
    u16t u16L;            // element specifier for accessing the high u16
    #endif
  } s32;                  // element spec. for acc. struct with low or high u16
} nt32;
#endif
```